Working With Databases

Introduction

With the LiveCode Database library, your application can communicate with external SQL databases. You can get data from single-user and multi-user databases, update data in them, get information about the database structure, and display data from the database in your stack. For a discussion of when it is appropriate to use an external database with LiveCode, see the topic When to Use a Database in the *LiveCode Script* guide.

This guide discusses how to install necessary software to communicate with databases, and how to use the Database library to communicate between LiveCode and a database.

This topic does not include discussion of how to set up and create a SQL database, which is beyond the scope of the LiveCode documentation.

To fully understand this topic, you should know how to write short scripts and should understand the basic concepts of SQL databases (rows and columns, database cursors, and SQL queries).

A few terms used in this topic, such as "field" and "cursor", are part of the standard terminology for working with databases, but have a different meaning in the context of LiveCode development. When going back and forth between database work and more general application development, be sure you understand which meaning is applicable in the context you're currently working in. When referring to database-specific terms, the documentation usually uses phrases like "database field" or "database cursor" to remind you of the context. See the Glossary within the product documentation for definitions of any term you are unsure of.

Introduction to Database Access

A database is an external resource that holds information, structured in a special form for quick access and retrieval. Databases can be:

- any size from small to extremely large
- located on the same system as the application or on a remote server
- accessible by one user at a time or by many users at once

SQL Databases

A SQL database is a database that you access and control using SQL, a standard database-access language which is widely supported. You use SQL queries (statements in the SQL language) to specify the part of the database you want to work with, to get data, or to make changes to the database.

LiveCode's database access is fully-featured. You can send any SQL statement to a database. You can open multiple databases (or multiple connections to the same database), maintain multiple record sets (database cursors) per connection, and send and receive binary data as well as text. You can do all this using the commands and functions in the *Database library*.

To see a list of LiveCode terms in the Database library, open the *Dictionary*, and type "*database*" into the search filter field.

The Basics of Database Structure

A database is built of records, which in turn are built out of database fields. A field is the smallest part of a database that can be separately addressed. Each database field contains a particular kind of information. This might be a name, a file path, a picture, or any other kind of information. Each record contains one value for each of its fields. A set of records is called a database table, and one or more tables comprise a database.

Here's an example: suppose you have a database of customers for your business. The fields of this database might include the customer name, a unique customer ID number, and shipping address. Each record consists of the information for a single customer, so each record has a different customer name, shipping address, and so on.

> **Note:** You may have noticed that the database structure being described resembles a multiple-card stack that has the same fields on each card. A database field is like a field in a stack, and a record is like a card. A stack set up this way can act as a database, in fact, but lacks some of the features of an external database, such as the ability to perform SQL queries and the ability to perform robustly when accessed by more than one user.

You can also think of the set of customer records as a grid (like a spreadsheet). Each row is a record, and each column is a field, so each cell in the grid contains a different piece of information about a particular customer. Here's an example:

ID	Customer Name	Address	Country
123	Jane Jones	234 E. Street	U.K.
836	Acme Corporation	PO Box 23788	USA
823	CanCo, Inc.	1 CanCo Blvd.	Japan

Figure 59 – Example Database Grid

There are three rows in this grid (each is the record for a particular customer) and four columns (each is one of the fields).

A row of the database means one single customer record, which has one value for each field. A column of the database means the set of all values for one of the fields, one for each record (for example, the set of all customer addresses).

More generally, each row describes one of the things in the database, and each column describes a particular state for each thing in the database.

The set of all customer records makes a table. Your database might include only this table, or it might include other related tables, such as a list of all sales. You can also connect related data from different tables of the same database. For example, if each record in the Sales table includes the ID number of the customer who bought the product, you can link all the sales records for a customer to that customer's record in the Customers table.

SQL and Record Sets – Database Cursors

SQL works primarily with sets of records, rather than individual rows. When you send a SQL query to a database, the query typically selects certain records which you can perform further operations on. The set of records resulting from a SQL query is called a database cursor, or record set. SQL queries let you describe the characteristics of the records you require, instead of processing each record one by one to find out whether it matches your criteria.

For example, consider the customer table we talked about in the previous section. To work with only US customers, you can write a SQL query that selects only records where the country field is "USA". This subset of records then becomes a record set. You can find out more about the fields and records contained in this record set, and move from record to record within this subset of the database. You can create more than one record set to work with more than one set of records at a time.

> **Note:** Different database implementations have different limitations on movement within a record set. For example, some databases won't let you move backward within a record set: instead, you must start at the first record and move forward in order to examine the data.

Choosing a Database

LiveCode directly supports the following database implementations:

- MySQL
- SQLite
- PostgreSQL

LiveCode also supports connecting to a database via ODBC. You can use ODBC to use Access, FileMaker, MS SQL Server and many other database implementations. See below for more information about ODBC.

LiveCode's database commands and functions use the same syntax regardless of what type of database you are connecting to. You don't need to learn a separate database language for each type. Instead, when you first open a database with the **revOpenDatabase** function, you specify the type as one of the parameters so LiveCode knows what type of database it's dealing with. The Database library handles the details of each type behind the scenes for you.

Reasons to Choose a Database Type

Which type of database to choose depends on a number of factors. If you need to work with an existing database, or already have a database manager installed, the decision is made for you. Likewise, if you're already an expert at a particular database implementation, you'll probably prefer to go on using that one.

Other factors in your choice may include price, performance, licensing model (commercial or open source), and platform support. If your users are on a particular set of platforms, you'll need to choose a database that is supported on all those platforms. Runtime do not endorse any particular database type, the information provided on the different types is for informational purposes only.

Overview of ODBC

Open Database Connectivity (ODBC) is a system that allows developers to access any type of compatible database in a standard way.

To communicate with a database, you usually have to add code that uses the database's own proprietary protocols. Without ODBC, in order to create a program that can communicate with--for example--FileMaker, Access, and Oracle databases, LiveCode would have to include code for three different database protocols. With ODBC, LiveCode can communicate with any of these database types using the same code.

ODBC Managers

To work with databases through ODBC, you need two pieces of software: an ODBC manager, plus a database driver for the specific database type you're using.

Windows and OS X include ODBC software with the operating system. For Linux systems, you can download an ODBC manager and a set of drivers (see the section below titled "Software for Database Access" for more information).

Performance for Direct Access Versus ODBC Access

Typically, accessing a database via ODBC takes more configuration and is slower than accessing the database directly. For this reason, LiveCode provides the ability to access MySQL, PostgreSQL databases directly without going through the ODBC protocol. This ability will be valuable for anyone doing complex or extensive professional database work.

The syntax of the functions in the Database library is identical for all database types, so you do not need to rewrite your scripts to take advantage of the increased efficiency of direct access.

Software for Database Access

To provide connectivity to databases, LiveCode works with database drivers--software that translates application requests into the protocol required by a specific database.

Finding Database Drivers

Database drivers for certain database types are included in the LiveCode distribution. (The list of included database types depends on the platform.) The sections below list which database drivers are included with which platforms.

If you have installed LiveCode, you have all the software needed to use the included database types. For other database types, you will need to obtain the appropriate database drivers before you can work with those databases.

> **Important:** This section includes links to third-party web sites and contains information about third-party software. This information is provided for your convenience, but LiveCode is not responsible for the software packages and sites referenced. Runtime regrets that it cannot provide any support for installation and configuration of any databasae.

MySQL

MySQL database drivers are included as part of the LiveCode installation on Linux, Mac OS X, and Windows systems.

PostgreSQL

A PostgreSQL database driver is included as part of the LiveCode installation on Linux, Mac OS X and Windows systems.

SQLite

Drivers for accessing this database are included with LiveCode. No additional installation is necessary.

ODBC managers and database drivers

To use a database via ODBC, you must install the necessary ODBC software for your platform. (Some operating systems include an ODBC installation.) ODBC software includes one or more database drivers, plus an ODBC manager utility.

ODBC on Windows systems

Windows systems include the MDAC (Microsoft Data Access Components) package as part of the standard system installation. To configure ODBC on Windows systems, use the ODBC Data Sources control panel.

ODBC on Mac OS X systems

OS X includes iODBC software as part of the standard system installation. To configure ODBC on OS X systems, use the ODBC Administrator application in the Utilities folder.

ODBC on Linux systems

LiveCode supports iODBC and UnixODBC on Linux systems. You can download the iODBC software from the iODBC web site at *http://www.iodbc.org/*. You can download the unixODBC software from the unixODBC web site at *http://www.unixodbc.org*.

Creating a DSN for ODBC access

Once you have installed the necessary software, you use the ODBC manager to create a DSN, which is a specification that identifies a particular database.

You use the DSN to connect to the database via ODBC. The following example opens a connection to a database whose DSN is named "myDB":

```
get revOpenDatabase("ODBC","myDB",,"jones","pass")
```

One of the advantages of setting up a DSN is that if you wish to change the location of the database, you only have to edit the DSN settings, not your application code. You can think of a DSN as a kind of shortcut or alias to your database.